12

Fourth SemiAnnual Technical Report

TRANSFORMATION of ADA PROGRAM UNITS INTO SILICON

UTEC-83-075

83 Apr 1 - 83 Nov 15

TECHNICAL REPORT

Department of Computer Science

University of Utah
Salt Lake City, Utah

12

Fourth SemiAnnual Technical Report

TRANSFORMATION of ADA PROGRAM UNITS INTO SILICON

UTEC-83-075

83 Apr 1 - 83 Nov 15

Elliott I. Organick, Principal Investigator
(801) 581-6087
organick@utah-20

Contractor: The University of Utah

Date of Contract: 81 SEPT 1

Expiring: 83 DEC 31

Sponsored by

Defense Advanced Research Projects Agency (DoD)
ARPA Order No. 4305

Under Contract No. MDA 903-81-C-0411, issued by
Defense Supply Service - Washington, DC 20310

November 1983

Table of Contents

Abstract for this Report 1
Recent Papers and Reports of this Project 3
1 Extended Summaries, by Area 6
1.1 Demonstrating how an Ada program unit is mapped to NMOS and connected and run with other Ada program units. 6
1.2 Transformation Research for Very High Level Silicon Compilers 7
1.3 PPL Circuit Arrays and Related Research 7
1.4 Signal Processing Applications 8
1.5 Rediflow Progress Report 10
2 Abstracts of Recent Papers 13
2.1 First Experiment Paper -- Abstract 13
2.2 Transformation Methodology -- Abstracts 14
2.3 VLSI Circuits and Design Aids using PPLs -- Abstracts 15
2.4 Rediflow and Prolog 18
3 Minipaper: System Modeling, Ada, and Signal Processing Applications in Silicon 20
3.1 Preface 20
3.2 Introduction 20
3.3 Modeling Signal Processing Subsystems in Ada 22
3.4 Testbedding 23
3.5 Using Ada to model systems for silicon compilation: Some Concluding Remarks 23
4 Cumulative Project Bibliography 25

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER UTEC-83-075	2. GOVT ACCESSION NO. AD-A141004	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) TRANSFORMATION of ADA PROGRAM UNITS INTO SILICON Fourth (and final) Semiannual Technical Report		5. TYPE OF REPORT & PERIOD COVERED 83 April 1 - 83 Nov 15
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) E. I. Organick et.al.		8. CONTRACT OR GRANT NUMBER(s) M DA-903-81-C-0411
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Utah Computer Science Department Salt Lake City, Utah 84112		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1001/1122
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Project Agency (DoD) 1400 Wilson Boulevard Washington, D.C. 20310		12. REPORT DATE 83 NOVEMBER
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Defense Supply Service Washington Rm 1d-245, The Pentagon Washington, D.C. 20310		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada-to-silicon transformation, high-level testing of Ada circuit equivalents path programmable logic, PPL design research, Read-Init-Parameter task in MUS, RIP chip design and testing, self-timed logic circuits, signal processing applications in silicon, silicon compiler, synthesizing VLSI circuits, VLSI circuits from Ada specifications

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report, augmented with several appended papers and supplementary reports, describes the most recent six months of work in the research project. "Transformation of Ada Programs into Silicon". This report is also the last of the series to be rendered under the current contract.

Our research has centered on methodologies for synthesizing asynchronous (speed-independent) VLSI circuits from Ada-like high-order specifications and

a companion development of a comprehensive strategy and means for simulating and testing the derived circuits in a manner similar to the testing of software modules. The relationship between our emphasis on speed-independent circuits and Ada (or Ada-like) specifications is not accidental, as Ada semantics for intermodule communication are mapped with relative ease, in meaning-preserving ways, into protocols useful for implementing speed-independent circuits. Also reported is work in areas of research perceived as related to our main theme, e.g., development of the applicative multiprocessing architecture named Rediflow. Such work has been partially seeded via this project.

Progress and new work is reported in five areas:

1. Work on the principal case study of this project: demonstrating by experiment how an Ada program unit is mapped to NMOS and connected and run with other Ada program units. Note that, in earlier reports, we referred to this effort under the heading: Converting portions of the DoD Internet Protocol to silicon.

2. High-level-language-to-silicon transformation research, including the theory and structure of comprehensive silicon compilers with high order language specifications as inputs.

3. Research and development on the design, fabrication, and testing of PPL circuit arrays with applications including the design of new styles of self-timed arithmetic units.

4. Launching of research into the application of our Ada-to-Silicon to Signal Processing computations, such as linear digital filters and DFT subsystems.

5. The definitional study for the Rediflow architecture, launched during the preceding reporting period has been greatly extended using funds in part made available through this project.

The body of the report provides summaries of (in one case a minipaper on) the work in each of these areas. In cases where work has led to completed papers, we present copies of their abstracts or other introductory material extracted from these papers. A cumulative list of our reports to date is given at the end of the report.

Abstract for this Report

This report, augmented with several appended papers and supplementary reports, describes the most recent six months of work on the research project, Transformation of Ada Programs into Silicon. This report is also the last of the series to be rendered under the current contract.

~~Our~~ research has centered on methodologies for synthesizing asynchronous (speed-independent) VLSI circuits from Ada-like high-order specifications and a companion development of a comprehensive strategy and means for simulating and testing the derived circuits in a manner similar to the testing of software modules. The relationship between our emphasis on speed-independent circuits and Ada (or Ada-like) specifications is not accidental, as Ada semantics for intermodule communication are mapped with relative ease, in meaning-preserving ways, into protocols useful for implementing speed-independent circuits. Also reported is work in areas of research perceived as related to our main theme, e.g., development of the applicative multiprocessing architecture named Rediflow. Such work has been partially seeded via this project.

Progress and new work is reported in five areas:

1. Work on the principal case study of this project: demonstrating by experiment how an Ada program unit is mapped to NMOS and connected and run with other Ada program units. Note that, in earlier reports, we referred to this effort under the heading: Converting portions of the DoD Internet Protocol to silicon.) [Responsible faculty: Organick, Lindstrom, Subrahmanyam, Smith.]

2. High-level-language-to-silicon transformation research, including the theory and structure of comprehensive silicon compilers with high order language specifications as inputs. [Responsible faculty: Subrahmanyam]

3. Research and development on the design, fabrication, and testing of PPL circuit arrays with applications including the design of new styles of self-timed arithmetic units. [Responsible faculty: Smith]

4. Launching of research into the application of our Ada-to-Silicon to Signal Processing computations, such as linear digital filters and DFT subsystems. [Responsible faculty: Organick, Henderson, Lindstrom, Subrahmanyam].

5. The definitional study for the Rediflow architecture, launched during the preceding reporting period has been greatly extended using funds in part made available through this project. [Responsible faculty: Keller, Lindstrom]

The body of the report provides summaries of (in one case a minipaper on) the work in each of these areas. In cases where work has led to completed papers, we present copies of their abstracts or other introductory material extracted from these papers. A cumulative list of our reports to date is given at the end of the report.

Recent Papers and Reports of this Project

The following is a list of the recent papers and reports completed during the current reporting period.

1. T.M. Carter, A. Davis, G. Lindstrom D. Klass, M.P. Maloney, B.E. Nelson, E.I. Organick, and K.F. Smith, "Transforming an Ada Program Unit to Silicon and Verifying its Behavior in an Ada Environment: A First Experiment", September, 1983, unpublished paper, a version of which is to be presented at CompCon84 in San Francisco.

2.

S. Purushothaman and P.A. Subrahmanyam, "An Algebraic Basis for Specifying and Reasoning about Protocols For Designing Self Timed Circuits", in VLSI 83, Edited by F. Anceau, North Holland, 1983.

S.V. Rajopadhye and P.A. Subrahmanyam, "TRACIS: Transformations on Ada for Circuit Synthesis. A Report on the methodology for a Silicon Compiler". Technical Report UTEC-83-003.

P.A. Subrahmanyam and I.V. Ramakrishnan, "Mapping Cube Graphs onto Processor Arrays", July 1983". Manuscript.

P.A. Subrahmanyam, "Synthesizing VLSI Circuits from Behavioral Specifications: A Very High Level Silicon Compiler and its Theoretical Basis", in VLSI 83, Edited by F. Anceau, North Holland, 1983.

P.A. Subrahmanyam, "Overview of a Conceptual and Formal Basis for An Automatable High Level Design Paradigm for Integrated Systems", 1983 IEEE International Conference for Computer Design and VLSI, New York, October 1983.

S.V. Rajopadhye and P.A. Subrahmanyam, "Formal Semantics for a Symbolic IC Design Technique". 1983 International Conference on Computer Design and VLSI, New York, October, 1983.

3.

K.F. Smith, B.E. Nelson, T.M. Carter, A.B. Hayes, "Computer-Aided Design of Integrated Circuits Using" Path-Programmable Logic", presented at Electro83 Conference, New York, NY, April 19, 1983

B.E. Nelson, K.F. Smith, T.M. Carter, "Cost Effective VLSI Design System", presented at IEEE 1983 International Symposium on Circuits and Systems, Newport Beach, Cal May 3, 1983.

T.M. Carter, "Automatic Implementation of Self-Timed State Machines in Integrated Circuits", presented at MIDCON83 Chicago, Ill, September 13, 1983.

J.C. Peterson and K.F. Smith, "Techniques for Automatic Layout of Constrained PPL Circuits", presented at IEEE International Conference on Computer-Aided Design ICCAD-83, Santa Clara, Cal., September 12-14. (No abstract)

K.F. Smith, "Design of Regular Arrays Using CMOS in PPL", presented at 1983 IEEE International Conference on Computer Design/VLSI in Computers ICCD '83, Port Chester, NY, November 1, 1983.

T.M. Carter and L.A. Hollaar, "The Implementation of a Radix-16 Digit-Slice Using a Cellular VLSI Technique", presented at 1983 IEEE International Conference on Computer Design/VLSI in Computers ICCD '83, Port Chester, NY, November 3, 1983.

K.F. Smith, B.E. Nelson, K.S. Stevens, "Student-Designed VLSI at the University of Utah", to be presented at IEEE 2nd Educational CAD/CAM Meeting, Washington, D.C., December 1983.

B.E. Nelson, "A Layered Database System Approach to Integrated Circuit Design in a Cell-Based Environment", PhD Proposal.

Carter, T.M., "Structured Arithmetic Tiling of Integrated Circuits", PhD Dissertation, University of Utah, Department of Computer Science, Dec., 1983.

4. None yet, but see Section3 of this report.

5.

R.M. Keller and F. Lin, "The Rediflow Simulator", Department of Computer Science, University of Utah, Sept. 1983 (Manual).

R. M. Keller, "Rediflow Multiprocessing", to be presented at CompCon84.

F. Lin, "A Distributed Load Balancing Mechanism for Applicative Systems", University of Utah, Department of Computer Science, 1983. Unpublished.

G. Lindstrom and P. Panangaden, "Stream-Based Execution of Logic Programs", to be presented at 1984 International Symposium on Logic Programming, Feb., Atlantic City.

P. Mishra and R. M. Keller, "Abstract Interpretation of Applicative Programs", to be presented at Eleventh Annual Symposium on Principles of Programming Languages", Jan., 1984, Salt Lake City.

P. Mishra, "Polymorphic Type Inference in Prolog", to be presented at 1984 International Symposium on Logic Programming, Feb., Atlantic City.

U.S. Reddy, "Transforming Logic Programs into Functional Programs", to be presented at 1984 International Symposium on Logic Programming, Feb., Atlantic City.

P.A. Subrahmanyam and J-H. You, "FUNLOG = Functions + Logic: A Computational Model Integrating Logic Programming and Functional Programming", Technical Report, UTEC 83-040, May 1983.

P.A. Subrahmanyam and J-H. You, "Pattern Driven Lazy Reduction: A Unifying Evaluation Mechanism for Functional and Logic Programs", to be presented at Eleventh Annual ACM Symposium on Principles of Programming Languages, Jan., 1984, Salt Lake City.

P.A. Subrahmanyam and J-H. You, "Conceptual Basis and Evaluation Strategies for a Programming Language Integrating Functional and Logic Programming", to be presented at 1984 International Symposium on Logic Programming, Feb., Atlantic City.

J. Tanaka and R. M. Keller, "S-code Extension in Function-Equation Language: User's Manual", Department of Computer Science, University of Utah, Unpublished memorandum, 1983.

J. Tanaka, "Optimized Concurrent Execution of an Applicative Language", Department of Computer Science, Univeristy of Utah, PhD Thesis, 1983. To appear.

1 Extended Summaries, by Area

1.1 Demonstrating how an Ada program unit is mapped to NMOS and connected and run with other Ada program units.

Participants: T.M. Carter, A. Davis, A.B. Hayes, G. Lindstrom, D. Klass, M.P. Maloney, B.E. Nelson, E.I. Organick, and K.F. Smith

We tested a chip representing a small but interesting component of the DoD Internet Protocol, large portions of which had been specified in Ada. The chip, which we have dubbed the RIP (for read initial parameters) is specified as an Ada package. It was transformed into a circuit that was fabricated through MOSIS. It was tested it at the electrical, gate, RTL, and Ada levels. Ada-level testing means that we have connected the chip to an Ada driver program executing in the host tester (an Intel 432 development system.) The (software and hardware) link between the driver and the chip is what we refer to as a semantically transparent interface. The succesful completion of this experiment (two years in the making) not only represents a new approach for development of in-system evaluation of system components. We have also refined and demonstrated new strategies and methodologies for building and testing subsystems having components exhibiting a mix of physical representation (heterogeneous system building).

The first version of the RIP chip was returned from MOSIS on 1 Aug. It was tested at both the electrical and gate levels and found to be almost perfect. The same chip was then connected into the Intel 432 peripheral interface and, on 30 Aug, tested successully at the Ada level. This experiment represents, to our knowledge, the first time anyone has demonstrated an executing Ada program "talking to" an Ada chip. The various software design and testing aids developed at Utah and previously reported were utilized in the above experiment. To successfully complete this experiment we also had to complete testing of our Intel 432 Cross Development System and to develop appropriate I/O Peripheral Subsystem software. A more complete "story" on this experiment is contained in [1] the abstract for which is given in the next section.

1.2 Transformation Research for Very High Level Silicon Compilers

Participants: P.A. Subrahmanyam, S.V. Rajopadhye, S. Purushothoman

Recent work in the implementation and design of the transformation system has mainly addressed bottlenecks caused by the tools that are being used in the current version of the transformation system. A brief summary of the current status of our transformation system is inluded in this report.

Along theoretical directions, we have applied the theoretical framework introduced in [16] to developing a specialized model for PPLs to aid analysis and synthesis at this level. We are also investigating how the model at the PPL level can be used for synthesis of special purpose arithmetic units [12]. Additionally, we have shown how the basic algebraic techniques can be used in proving a generalized version of the Leiserson-Saxe retiming algorithms and in demonstrating the correctness of some further transformation on systolic designs. Some techniques for mapping certain classes of data flow graphs onto regular arrays suitable for VLSI implementation have also been investigated [13]; this class includes quite a few signal processing applications.

1.3 PPL Circuit Arrays and Related Research

Participants: K.F. Smith, T.M. Carter, B.E. Nelson

After receiving the first run of the RIP chip in August, we made a minor correction to its design and resubmitted it to MOSIS on the M39E run. Eight packaged circuits were returned approximately November 1. The first run of the RIP chip was tested using a Tektronix DAS system and the good parts were then used in the Ada testbed environment. Packaged circuits from the second run, however, were not tested using the DAS system but were immediately used in the Ada testbed. They were found not to operate correctly in the testbed. Even the portions of the circuit that were not changed from the previous run were non-functional in this run. Both the previous and present runs of this circuit were fabricated at COMDIAL. It would appear that the problem is poor quality control in the fab line. We have resubmitted the circuits for another run.

During the past 10 months, we have submitted and received approximately 17 different circuits from MOSIS with the following results (two of these were versions of the RIP chip, one of these had a design error and the other was non-functional):

	Working	Design Errors	Non-Functional	Not Yet Checked
NMOS PPL	3	4	2	2 (rec'd 10/20)
CMOS	0	0	0	2 (rec'd 11/1)
NMOS CUSTOM	1	1	2	0

The non-functional circuits did not work due to poor processing quality. Of the 4 in this category, 3 were fabricated by COMDIAL and 1 by Hewlett-Packard.

The circuits with design errors were found to operate as designed but required design changes to operate as desired.

1.4 Signal Processing Applications

Participants: E.I. Organick, T.C. Henderson, J.W.L. Ogilvie, M.P. Maloney

We are now investigating a technique for using Ada as a system-modeling language to specify signal processing algorithms, such as linear digital filters and discrete fourier transforms, in fully-unfolded form. ["Modeling" is used here in the sense of specifying the complete static allocation of resources for the purpose of construction, as in the blueprints that one would prepare for any two-dimensional architectural layout (landscape, building, or circuit). This is in contrast to the use of "modeling" in specifications exhibiting higher levels of abstraction where expressing the static allocation of resources is neither critical nor even, in many case, appropriate.]

The fully-unfolded forms of these algorithms express as explicitly as is practical the net of processing elements comprising the ensemble of components of the subsystem. [Fully unfolded forms are those that explicitly express intended distributed computations.] In the unfolded form there is no sharing of data and control structures. Communication among the components follows asynchronous protocols consistent with Ada semantics. Thus, the ensembles of components correspond to mosaics of asynchronously communicating processing elements that exhibit high-levels of concurrency.

Mosaics are generalizations of systolic arrays in which each tile of the mosaic is specified as an instantiated Ada generic package object that includes a single (quite elementary) task. Mosaics differ from systolic arrays principally because we do not restrict the array (processing) elements to be of the same type.

In principle, mosaics can be transformed in systematic ways into equivalent circuitry and then set in silicon; we can use transformation techniques based on research reported elsewhere in this and preceding reports of this Project.

To apply the unfolding style advocated here for arbitrarily large applications, (e.g. filters and DFTs whose signal flow graphs have large numbers of nodes) we have found that Ada must either be modified or augmented with an Ada program generator. This need arises because standard Ada does not permit us to explicitly declare arrays of package elements such that an element can "know" which element (in the array) it is and address neighboring array (mosaic) elements in terms of this knowledge. We can identify three approaches, two to modify and one to augment Ada.

1. Modify Ada: either tasks must be "promoted" to compilation units, or a *superset of Ada would be needed* in which packages can be promoted to what amounts to "first class objects". Given the large effort that has already been invested in Ada compiler development, the latter approach appears to be more practical, although not without its shortcomings.

   At least one acceptable superset is available: the superset, known as 432 Ada.[1]

2. Augment Ada with a source program generator which would take as its input instructions to generate for each system under development, the desired mosaic of generic package instances; the output of the generator would then serve as input to the standard Ada compiler. This approach may well prove the most useful.

We anticipate that a byproduct of this research is a style of programming in Ada which facilitates silicon compilation.

[1] 432 Ada is Intel's version of Ada used for its iAPX 432 Systems. This superset permits us to use package access variables.

1.5 Rediflow Progress Report

by Robert M. Keller and G. Lindstrom

The goal of the Rediflow project continues to be the development of a programmable, scalable multiprocessing system based upon communication and data-structuring concepts from the realm of functional programming. In order to provide meaning for the items in our report, we present the following review of the Rediflow approach. (See [4] for more detail.)

While conceding that fine-grain parallel processing may be the best solution for problems with highly-regular and static structure, Rediflow aims at providing concurrent execution for irregularly-structured problems of medium and larger granularity which are not treatable by regular approaches (SIMD machines, processor arrays, array processors, etc.). Rediflow attempts to deal with the issue of mapping problems onto the machine by basing the computational model on the "graph reduction" approach, wherein the program and data are represented as a distributed expanding and shrinking graph, transmutations of which may concurrently take place at many processor sites. Compared to the process-oriented approach, reduction seems to provide a smaller grain, making automatic distribution more likely to be feasible. By equating units of granularity with function bodies, the grain is also sufficiently-large that communication penalties are not suffered from trivial operations. However, the efficiency of a pure reduction implementation is sub-optimal, due to the reliance by reduction on dynamic storage allocation for rather trivial computational operations. For this reason, Rediflow combines the work-distribution ability of the reduction model with the more storage-efficient von Neumann model, the latter being encapsulated into modules with an external "data flow" character.

As initially stated, programmability and scalability are primary concerns. Another facet of the Rediflow research is thus concerned with the physical distribution of units of work to processors. This would be trivial in shared-memory models of multiprocessors. However, scaling the latter lengthens processor-memory delays and introduces complicated mechanisms to insure memory coherence. Accordingly, we are pursuing a model in which every unit of memory is dedicated to some processor, but which uses a common logical address space to manage information across such memories and packet switching to effect the physical transfer of data on demand. The load distribution and balancing aspects of this model are achieved by a technique analogous to "fluid flow" over a "surface" of processor-memories, with units of reduction work as molecules of the fluid. This aspect, along with initial research into reliability, are discussed in a dissertation proposal [5].

Effort has gone into three main areas:

1. Simulation of the Rediflow architecture.

2. Programming language issues.

3. Applications programming.

Most simulation effort has been put into enhancing the simulator for the Rediflow architecture [3]. Here we have installed code which combines sequential processes with the reduction model, a switching layer simulator, the load-distribution and balancing mechanism, and extensive instrumentation. As a result, the concepts concerning dynamic load distribution have been confirmed. Considerable study remains on how to best control distribution parameters. Also needed is a modification which provides accurate detailed times for each of the functions of the processor. The current model makes rather gross assumptions about these, although it does seem to realistically model communication delay in the switching layer.

The observations made thus far indicate that the switching layer is not a bottleneck, due to sufficient locality inherent in the granularity of Rediflow functions. It also confirms the existence of this locality by inter-node distances of messages. In a typical run, 75% of the data packets make fewer than 2 node hops, while upwards of 80% of the functions spawned are executed within two nodes of their parent. Speedups of greater than 10 have been obtained on a 16 processor model for pure divide and conquer applications, while speedups of 3 to 4 have been obtained for applications with a less pure mixture of sequential and concurrent code. Major improvements in overhead for the reduction model are now being sought to enable competition with more conventional execution strategies.

Current work on applications includes logic programming, searching, database querying and updating, signal processing, and or irregularly- or impredictably-structured numeric applications. As an example, multiple approaches in the evaluation of logic programs on

Rediflow are being pursued. One set of preliminary ideas given in [15]. A second is elaborated upon below.

In the programming language area, the FEL -> Rediflow compiler is being re-written to accommodate new ideas in optimization. Although the integrated sequential processes have been present in the Rediflow simulator for some time, we have not had an accompanying high-level driver in FEL. This is being remedied [17, 18]. Other aspects of optimization include a type inference mechanism, viewed as being essential for the generation of efficient production code [7].

Progress has also been made in designing an interpreter for a restricted class of logic programs, which is intended to exploit the reduction/dataflow execution model employed by Rediflow. Our initial conception of this interpreter is outlined in [6], the abstract for which is given in the next section.

Continuation efforts in this direction are proceeding, with the following near term activities planned:

1. Extension of the interpreter to accommodate a reasonably 'full' logic programming language (probably Prolog).

2. Optimization of the interpreter code through judicious use of Kahn processes.

3. Economization of the resulting search strategy through application of the Keller/Sleep applicative caching mechanisms [2].

4. Quantitative evaluation of the resulting implementation on the Rediflow simulator.

2 Abstracts of Recent Papers

2.1 First Experiment Paper -- Abstract

Transforming an Ada Program Unit to Silicon and Verifying its Behavior in an Ada Environment: A First Experiment

by

T.M. Carter, A. Davis, A.B. Hayes, G. Lindstrom, D. Klass, M.P. Maloney, B.E. Nelson, E.I. Organick, and K.F. Smith

An Ada package has been transformed into an NMOS chip and successfully "appended" to a larger Ada program executing on a general purpose computer (micromainframe). This demonstration is a component of a research effort to develop a very high-level silicon compiler that can convert Ada program units to silicon using software design aids.

Ada program units in the micromainframe "talk to" an Ada task embedded in the NMOS-package. The selected Ada package was used to specify the chip to assess the potential of Ada as a hardware specification (and *description*) language. The package specification was transformed to a silicon composite partly by automatic and partly by manual means. Further work is in progress to increase the degree of automation and to simplify the interface with the mainframe.

Although the demonstration circuit is a small one, the active area measures 3.8 by 3.0 mm and equivalent to just under 2000 2-input NAND gates, the specifying Ada package with over 100 lines of Ada code involves communication with tasks in three other Ada packages and has a moderately rich declarative and control structure. (The functionality of this particular chip is not impressive but does, however, represent a component of a larger interesting program, the DoD Internet Protocol, much of which was also specified in Ada.) The circuit was implemented with the PPL cell-based methodology; it is also the first circuit demonstrating the ASSASSIN silicon compiler. The circuit, fabricated through MOSIS, adheres to a completely asynchronous (speed-independent) design discipline, a well-accepted style for building very large reliable systems on silicon substrates from smaller building blocks adhering to similar programming and logic diciplines.

The paper outlines the procedures we have followed in conducting this experiment and

briefly mentions the tools we have developed and used, other planned design aids still to be developed, lessons learned so far, and some new ideas and approaches under current consideration or investigation.

2.2 Transformation Methodology -- Abstracts

TRACIS: Transformations on Ada for Circuit Synthesis:
A Report on the methodology for a Silicon Compiler

by

Sanjay V. Rajopadhye and P. A. Subrahmanyam

This report describes in detail, the ongoing design and implementation of a transformation system, for "compiling" specifications of integrated circuits into silicon. There are many levels in this process, and the area that we focus on produces target specifications of asynchronous and synchronous control units and the associated data paths. This target is compatible with the ASSASSIN system which generates layouts from specifications of control units. The input to our system is an Ada program (restricted to a single Procedure Body) which specifies a certain computation. The Procedure Body is itself assumed to contain no package or task declarations or inatantiations and no Entry call statements. The result of the transformations performed by the system is a program consisting of the original specifications, witn the target description appended to it.

Currently the system is in an experimental stage, and many of the intermediate decisions are specified interactively by the user. In spite of these limitations we feel that it is a valuable tool that we can use to study the exact mechanisms of the transformations as well as to understand how various syntactic and/or semantic analyses (e.g. data flow analysis) of the input can affect them.

Overview of a Conceptual and Formal Basis for
An Automatable High Level Design Paradigm for Integrated Systems

by

P.A. Subrahmanyam

(This is a summary of an invited talk presented at the 1983 International Conference for Computer Design and VLSI, Westchester).

It is important that design methodologies for special purpose VLSI circuits enable a smooth embedding of the resulting circuits into larger systems that consist of both software and specialized hardware components e.g., on-board control systems. Furthermore, we believe that it is advantageous if automated design aids that support such methodologies are based on a uniform set of principles -- ideally, on a unifying theoretical basis. In this paper we delineate a conceptual and theoretical basis intended to aid both in the analysis and in the mechanical synthesis and verification of special purpose VLSI systems, proceeding from high level behavioral specifications.

2.3 VLSI Circuits and Design Aids using PPLs -- Abstracts

Computer-Aided Design of Integrated Circuits Using Path-Programmable Logic

by

Kent F. Smith, Brent E. Nelson, Tony M. Carter, Alan B. Hayes

Electro83 Conference, New York, NY, April 19, 1983

The Path Programmable Logic (PPL) Design Methodology as developed at the University of Utah is described. Derived from the SLA, the PPL methodology reduces integrated circuit design times by an order of magnitude over custom design. The Utah Design Environment for PPL circuit design is also described. This system includes tools for PPL cell design, circuit layout using the defined PPL cells, simulation, placement and electrical checking, and pattern generator tape preparation. This system was used to design three small test chips which were fabricated through the MOSIS facility.

Cost Effective VLSI Design System

by

Brent E. Nelson, Kent F. Smith, Tony M. Carter

IEEE 1983 International Symposium on Circuits and Systems, Newport Beach, Cal May 3, 1983.

This paper describes a cost-effective system for the design, layout, and simulation of structured-logic integrated circuits. It is an alternative to high-cost graphical layout systems for VLSI because design is performed from an inexpensive, remote terminal. The system is implemented around a design methodology known as Path-Programmable Logic (PPL).

Automatic Implementation of Self-Timed State Machines in Integrated Circuits

by

Tony M. Carter

MIDCON83 Chicago, Ill, September 13, 1983

This paper presents a method and software system for automatically implementing self-timed state-machines in integrated circuits. The software system is called ASSASSIN. It compiles a linguistic, technology independent specification language into an integrated circuit composite by using a one-hot state encoding and an integrated circuit implementation methodology known as Path-Programmable Logic (PPL). It provides a text editor for composing the state-machine description, a simple functional simulator which allows the user to step through the operation of the machine being designed, and a compiler to PPL programs (which are currently implemented in the NMOS technology).

Design of Regular Arrays Using CMOS in PPL

by

Kent F. Smith

1983 IEEE International Conference on Computer Design/VLSI in Computers ICCD '83, Port Chester, NY, November 1, 1983.

The design of regular arrays using a methodology known as Path Programmable Logic (PPL) with a single layer metal, P well, CMOS process is discussed. PPL cell sets for both static and dynamic circuits using this process are presented. The static CMOS cells are similar to those cells previously designed for an NMOS process but the dynamic CMOS cells are a unique departure from these static designs. A simple counter circuit is used as an example to demonstrate the relationship between the present CMOS designs and previous NMOS designs.

The Implementation of a Radix-16 Digit-Slice Using a Cellular VLSI Technique

by

Tony M. Carter, Lee A. Hollaar

1983 IEEE International Conference on Computer Design/VLSI in Computers ICCD '83, Port Chester, NY, November 3, 1983.

The paper presents the design of a radix-16 digit-slice arithmetic processor using a structured tile approach to integrated circuit design. It treats the logic design of a set of arithmetic operators which conform to a theory of decomposition of structures for arithmetic, their NMOS implementation as tiles and the implementation of the digit-slice using these tiles.

Student-Designed VLSI at the University of Utah

by

Kent F. Smith, Brent E. Nelson, Kenneth S. Stevens

IEEE 2nd Educational CAD/CAM Meeting, *Washington, D.C., December 1983.*

The design of integrated circuits by students in the VLSI design course at the University of Utah is presented. Students are able to design 10,000 to 20,000 transistor circuits in a single quarter using a methodology known as Path Programmable Logic (PPL). An example of a typical student's project is included.

A Layered Database System Approach to Integrated Circuit Design in a Cell-Based Environment

(PhD Proposal)

by

Brent E. Nelson

A complete design system for Structured Tiling integrated circuits is developed. The use of a database management system as a foundation for a set of CAD tools is explored. Specifically, library management functions associated with the structured tiles are handled in a simple, efficient manner when a database system approach is used. Included is also the implementation of a number of structured tiles which extend Path Programmable Logic (PPL) to include datapath elements.

Structured Arithmetic Tiling of Integrated Circuits
(PhD Dissertation)

by

Tony M. Carter

This dissertation treats the development of an integrated circuit design methodology known as Structured Arithmetic Tiling. Structured Arithmetic Tiling employs Robertson's "Theory of Decomposition of Structures for Binary Addition and Subtraction" as a basis for an efficient integrated circuit design methodology for implementing arithmetic structures. This methodology is demonstrated through several simple examples and through the implementation of a complex arithmetic structure known as a Radix-16 Digit-Slice.

The implementation of Structured Arithmetic Tiling is built upon the foundation offered by the Structured Tiling integrated circuit layout methodology. Structured Tiling provides a formalized basis for the development of structured integrated circuit design methodologies. It is of sufficient generality to allow the implementation of many existing and proposed design methodologies.

2.4 Rediflow and Prolog

Stream-Based Execution of Logic Programs

by

Gray Lindstrom and Prakash Panangaden

We report a new execution model for logic programs, combining the following key features:

* a stream-based version of the "classical" backtracking execution model;
* OR-parallelism, with a particular form of induced AND-parallelism;
* an applicative formulation, except for indeterminate stream merging;
* concurrent processing of several top-level goals, if desired;
* a "pure code" utilization of program clauses, with all instantiation done via composition of substitution records, and

* exploitation of two complementary evaluation mechanisms for functional languages: reduction and dataflow.

The model is informally sketched, and then formally defined using the Function Equation Language FEL.

3 Minipaper: System Modeling, Ada, and Signal Processing Applications in Silicon

Signal Processing-to-Silicon Using Ada as a Hardware Specification Language: An Initial Investigation

by

E.I.Organick, T.C. Hendserson, M.P. Maloney, and J.W.L. Ogilvie

3.1 Preface

The following paper expands on a number of the ideas introduced in Section 1.4. A more complete, but as yet untested set of illustrations of the technique is given in a set of three preliminary notes [9], a revised (and hopefully publishable) form of wnich is now under development as part of a more complete version of this paper.

3.2 Introduction

We begin with the premise that our "business" is the building of heterogeneous systems whose components are implemented across a spectrum of technologies and levels of data abstraction. A signal processing system, for example might consist of a conventional host processor interfaced with one or more mosaic structures, each implemented in VLSI, and specialized, say, as linear digital filters, DFT processing subsystems, etc.

Fitting together the diverse components of such systems has always been a troublesome design and implementation challenge. A major stumbling block has been lack of a suitable system modeling language (SML) fo~~al enough in its syntax and semantics to be used for specifying system components at required levels of abstraction and for simulation at chosen levels of abstraction. Moreover, system components that one might decide to build rather than merely simulate could, in principle, then be built or compiled from its specification as formally expressed within the SML. Software engineers have been developing an appreciation of this approach to system building for some time. Nowadays, hardware engineers are beginning to acquire similar views [11, 10].

It is intriguing to consider what must be the required relationship between the conventional compiler, which can transform a high-order language specification of a system component into an instruction/data structure for a host-machine, and a silicon compiler which, were its development sufficiently advanced, could compile the same source-language specification into a semantically equivalent hardware component. An examination of this relationship has led us to the following observation (and principle): if we are to build heterogenous systems with the aid of compilers, then it must be true that all buildable parts of the system should be produced by compilers that are driven by the same language syntax and semantics. This principle further implies that conventional and silicon compilers must recognize the same compilation units of the SML; and, only these compilation units can be converted to silicon. By adhering to this principle, we can be assured that system parts built using different implementation media (and exhibiting different levels of data abstraction) will fit consistently at their interfaces. We also have the assurance that system parts can be replaced with semantically equivalent ones when implemented in different technologies. [For example, were the SML standard Ada, then naked Ada tasks, which are not legal Ada compilation units, could not be "extracted" from a program, converted into silicon, and the "hooked up with" the rest of the program executing, say, on a general-purpose host.]

Having in hand an appropriate SML, permits the engineer to view an entire system under design (an ensemble of interacting components) as a single "program" whose static components correspond to program units and whose associated interfaces have clearly specified semantics. Thus, when choosing a particular implementation medium for a selected component C, a guiding principle is preservation of C's behavior as specified in the SML. This implies as well the preservation of C's interface (semantics) with the other system components (call these "Others"), regardless of what medium of implementation has been chosen for Others. Thus, we have the derived principle that alternative interfaces between a pair of components may differ physically but not semantically. If the SML is rich enough, it should be possible to specify a sufficiently wide range of implementations for any selected system component or group of components, thus assuring the designer that he/she does not lose control, through lack of expressive power, of the design responsibility.

We are aware that SMLs that fully meet the above criteria may not now exist, even for use in designing a limited class of systems, e.g., systolic arrays. On the other hand, since we wish to proceed as far as possible with the development of a system building methodology and style that is based on use of an SML, we are obligated to seek an existing modeling language whose characteristics sufficiently approximate the desired SML. For this reason we have chosen to focus on Ada: to learn how to use it as an SML for building systems of heterogenous components and, in the course of doing so, to learn what crucial features if any it now lacks. In retrospect, we in the Ada-to-Silicon group have, for several years, been studying Ada as a candidate SML; moreover, we have already reported on one experiment in which Ada has been used as an SML (the RIP chip).

3.3 Modeling Signal Processing Subsystems in Ada

Essentially all signal processing subsystems can be modeled with dataflow graph semantics. Such graphs can in turn be modeled using ensembles of Ada tasks (representing graph nodes) and asynchronous intertask communication (representing arcs joining these nodes). Thus, to the extent it is useful, one can assure for a signal processing application a one-one correspondence between an Ada program structure and a dataflow structure at the level of granularity where nodes are represented by tasks or, more precisely, as packages containing these tasks. One can also ensure that there is no sharing of data between and among such Ada tasks except via explicit message-based communication. Nor is there need to have contention for the service offered by the tasks, i.e., where two or more requestor tasks would try to deposit requests in the same queue.

An important simplification, related to the applicative nature of signal processing, is that communication channels needn't be implemented by mapping into circuitry the general, but relatively complex, Ada rendezvous semantics. Instead, these channels may be correctly modeled by having the tasks described in the preceding paragraph perform data transmission by using simpler Send (Receive) operators which put (take) packets into (from) queues, causing blocking of a requestor task when a queue is full (empty). The use of these operators and the specific queues to which they apply are easily specified in Ada; for each queue, one defines a package whose operators, Send and Receive, are implemented in terms of local server task that manages a local fifo queue. We note that the hardware logic design equivalent for such a queue package (object) is straightforward as is the simple-to-understand-and-implement request/acknowledge protocols required to implement the Send (Receive) operator calls.

In this regard, Ada appears to be an attractive starting point specification language for the signal processing domain. In particular, with respect to their interface characteristics, tasks in the corresponding Ada model (nodes of the signal processing model) are all equally attractive candidates for conversion to silicon. In short, interesting signal processing applications are in our view initially modeled as dataflow graphs consisting of a number of nodes, where nodes are themselves modeled by Ada tasks embedded in packages.

3.4 Testbedding

Let us now suppose we are successful in developing a uniform methodology for converting any signal processing node to silicon and for integrating it into its local environment. Further, assume this methodology is also made viable through use of a common set of design aids and test stations. Then, in principle, subsets of the Ada tasks for a signal processing application may be selected for conversion to silicon and integrated into the rest of the processing subsystem executing on a conventional host. Note that this conversion can be done incrementally (i.e., one task after another within a subset) until a sufficient number of related tasks have been so converted. Alternatively, tasks can be independently (i.e., "concurrently") converted to silicon; in this mode, different teams of designers would work cooperatively ("in parallel") on different parts of the overall conversion effort.

3.5 Using Ada to model systems for silicon compilation: Some Concluding Remarks

What, in summary, can we say is the essence of the system modeling "art" as discussed above? We present this art as a tentative set of modeling principles to be adhered to when using Ada to model systems for later translation into silicon.

1. We must somehow choose a 1-1 correspondence between Ada compilation units we supply to an Ada compiler and those we want to represent as distinct physical (silicon) units.

2. The particular distribution of computation among the physical units (and also between the components to be reduced to silicon and the remaining compilation units in the specifying program) must be deducible by an Ada silicon compiler.

3. When we intend to construct arrays, or other networks of physical units, we must make explicit the communication between the corresponding Ada compilation units (by using procedure and task entry calls that properly identify their targets). The purpose here is to specify the topology in which the corresponding physical units are to lie.

4. When we wish to prohibit sharing of communication channels among physical units to achieve maximum concurrency and minimize arbitration circuitry, we must avoid writing Ada specifications from which such sharing can be inferred.

5. Memory should be distributed among the Ada units, just as it will be among their physical counterparts. This means, for example, that pointers into a central memory must not be passed within communication packets. Thus, actual values (e.g., records of type complex number, in the case of the FFT) are passed, but not pointers to them.

6. We must restrict recursion to occur, if at all, to within (and only within) single Ada compilation units.

This list of principles may not be complete and some of its elements may not even be necessary. Further research along these lines is underway.

4 Cumulative Project Bibliography

This section contains a cumulative list of the papers, reports, and theses regarded as direct, or indirect "products" of this Project.

[1] Carter, T.M.
ASSASSIN: An Assembly, Specification and Analysis System for Speed-Independent Control-Unit Design in Integrated Circuits Using PPL.
Master's thesis, University of Utah, Department of Computer Science, June, 1982.

[2] Carter, T.M.
ASSASSIN: A CAD System for Self-Timed Control-Unit Design.
Technical Report UTEC-82-101, University of Utah, October, 1982.

[3] Carter, T.M.
ULC: A System for Converting and Merging IC Layouts in Varying Representations.
Technical Report UTEC-83-029, Department of Computer Science, University of Utah, April, 1983.

[4] Carter, T.M., Davis, A., Hayes, A.B., Lindstrom, G., Klass, D., Maloney, M.P., Nelson, B.E., Organick, E.I. and Smith, K.F.
Transforming an Ada Program Unit into Silicon and Verifying its Behavior in an Ada Environment: A First Experiment.
September, 1983.

[5] Carter, T.M.
ASSASSIN: A CAD System for Self-Timed Control-Unit Design.
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. To appear.

[6] Carter, T.M.
Automatic Implementation of Self-Timed State Machines in Integrated Circuits.
Professional Program Session Record 4 of MIDCON 1983, Sept., 1983.

[7] Carter, T.M., and Hollaar, L.A.,
The Implementation of a Radix-16 Digit-Slice Using a Cellular VLSI Technique.
Proceedings of the 1983 IEEE International Conference on Computer Design/VLSI in Computers. Nov., 1983, pp 688-691.

[8] Carter, T.M.
Structured Arithmetic Tiling of Integrated Circuits
PhD Dissertation, University of Utah, Department of Computer Science, Dec., 1983.

[9] Drenan, L.A.
On Transforming Ada to Silicon.
Master's thesis, University of Utah, Department of Computer Science, August, 1982.

[10] Drenan, L.A., Organick, E.I.
Ada to Silicon Transformations: The Outline of a Method.
Technical Report UTEC-82-016, Dept. of Computer Science, University of Utah, Sept, 1982.

[11] A. B. Hayes.
Self Timed IC Designs with PPL's.
In Proceedings of the 1983 Cal Tech VLSI Conference, pages 257-274. Caltech, Computer Science Press, March, 1983.

[12] R.M. Keller and F. Lin.
The Rediflow Simulator
1983.

[13] R.M. Keller.
Rediflow Multiprocessing.
In Compcon '84. IEEE, 1984.

[14] Lew, K.O., Organick, E.I.
Mapping Ada Compound Statements into Distributed Self-Timed Logic Circuits.
Technical Report UTEC-83-028, Department of Computer Science, University of Utah, November, 1982.

[15] F. Lin.
A Distributed Load Balancing Mechanism for Applicative Systems.
1983.

[16] Lindstrom, G., Organick, E.I., Klass, D., Maloney, M.
Ada Specifications for the DoD Internet Protocol: The INM_OUT Submodule, Report No. 1.
Technical Report, Department of Computer Science, University of Utah, November, 1982.

[17] Lindstrom, G. and Panangaden, P.
Stream-Based Execution of Logic Programs.

To appear in Proc. 1984 Int'l. Symp. on Logic Programming, Atlantic City, Feb. 1984.

[18] P. Mishra and R. M. Keller.
Abstract interpretation of Applicative programs.
In Eleventh Annual Symposium on Principles of Programming Languages. ACM, January, 1984.

[19] P. Mishra.
Polymorphic Type Inference in Prolog.
In 1984 International Symposium on Logic Programming. IEEE, February, 1984.

[20] Nelson, B.E.
ASYLIM User's Manual
Dept. of Computer Science, Univ. of Utah, 1982.

[21] Nelson, B.E.
ASYLIM: A Simulation and Placement Checking System for Path-Programmable Logic Integrated Circuits.
Master's thesis, University of Utah, Department of Computer Science, October, 1982.

[22] Nelson, B.E., Smith, K.F., Carter, T.M.
Cost Effective VLSI Design System.
In Proc. IEEE Int'l. Symp. on Circuits and Systems. May, 1983.

[23] Nelson, B.E.,
A Layered Database System Approach to Integrated Circuit Design in a Cell-Based Environment.
PhD Proposal, Dept. of Computer Science, University of Utah.

[24] Organick, E.I., and Lindstrom, G.
Mapping high-order language units into VLSI structures.
In Proc. COMPCON 82, pages 15-18. IEEE, Feb., 1982.

[25] Organick, E.I., Maloney, M., Klass, D, Lindstrom, G.
Transparent Interface Between Software and Hardware Versions of Ada Compilation Units.
Technical Report UTEC-83-030, Department of Computer Science, University of Utah, April, 1983.

[26] Organick, E.I., Carter, T., Lindstrom, G., Smith, K. F., Subrahmanyam, P.A.
Transformation of Ada Programs into Silicon. SemiAnnual Technical Report.

Technical Report UTEC-82-020, Dept. of Computer Science, University of Utah, March, 1982.

[27] Organick, E.I., Carter, T.M., Hayes, A.B., Nelson, B.E., Lindstrom, G., Smith, K., Subrahmanyam, P.A.
Transformation of Ada Programs into Silicon. Second SemiAnnual Technical Report.
Technical Report UTEC-82-103, Dept. of Computer Science, University of Utah, November, 1982.

[28] Organick, E.I., Keller, R.M., Lindstrom, G., Smith, K.F., Subrahmanyam, P.A., Carter, T., Klass, D., Lew, K.O., Maloney, M.P., Nelson, B.E., Purushothaman, S., Rajopadhye, S.V.
Transformation of Ada Programs into Silicon. Third SemiAnnual Technical Report.
Technical Report UTEC-83-026, Dept. of Computer Science, University of Utah, April, 1983.

[29] Peterson, J.C. and Smith, K.F.
Techniques for Automatic Layout of Constrained PPL Circuits.
Proceedings of the 1983 International Conference on Computer-Aided Design. Sept., 1983, pp 194-195.

[30] Purushothaman S., and Subrahmanyam, P.A.
An Algebraic Model of Seitz's Weak Conditions for Self Timed Systems.
Technical Report UTEC-82-066, Department of Computer Science, University of Utah, October, 1982.

[31] Rajopadhye, S. V. and Subrahmanyam, P.A.
TRACIS: Transformations on Ada for Circuit Synthesis.
Technical Report UTEC-83-003, University of Utah, Dept. of Comp. Sci., August, 1983.

[32] Rajopadhye, S.V. and Subrahmanyam, P.A.
Formal Semantics for a Symbolic IC Design Technique.
Paper presented at the 1983 International Conference on Computer Design and VLSI, New York, November, 1983.

[33] Ramachandran, R.
A Complexity Computation Package for Data Type Implementations.
Master's thesis, University of Utah, Department of Computer Science, June, 1982.

[34] Ramakrishnan, I.V. and Subrahmanyam, P.A.
On Mapping Hypercube Graphs onto Linear Systolic Arrays.
Technical Report CS-83-203, Department of Computer Science, University

of Utah, March, 1983.

[35] Reddy, U.S.
Transforming Logic Programs into Functional Programs.
In 1984 International Symposium on Logic Programming. IEEE, February, 1984.

[36] Smith, K.F., Nelson, B.E., Carter, T.M., Hayes, A.B.
Computer-Aided Design of Integrated Circuits Using Path Programmable Logic.
IEEE Electro 83 Professional Program Session Record, New York" April, 1983.

[37] Smith, K.F., et al.
PPL User's Manual.
Internal Report, Dept. of Computer Science, University of Utah, July, 1982.

[38] Smith, K.F.
Design of Regular Arrays Using CMOS in PPL.
Proceedings of the 1983 IEEE International Conference on Computer Design/VLSI in Computers, Nov., 1983, pp 158-161.

[39] Smith, K.F., Nelson, B.E., Stevens, K.S.,
Student-Designed VLSI at the University of Utah.
To be presented at IEEE 2nd Educational CAD/CAM Meeting, Washington, D.C., December 1983.

[40] Subrahmanyam, P.A.
Abstractions to Silicon: A New Design Paradigm for Special Purpose VLSI Systems.
Technical Report UTEC-82-065, University of Utah, January, 1981.
Revised May, 1982.

[41] Subrahmanyam, P.A.
From Anna+ to Ada: Automating the Synthesis of Ada Package and Task Bodies.
Technical Report Internal Report, Dept. of Computer Science, University of Utah, March, 1982.

[42] Subrahmanyam, P.A.
Language Issues in Transformation Systems
Technical Report UTEC-82-069, University of Utah, November, 1982.

[43] Subrahmanyam, P.A. and Rajopadhye, S.

Automated Design of VLSI Architectures: Some Preliminary Explorations.
Technical Report UTEC-82-067, University of Utah, October (Revised), 1982.

[44] Subrahmanyam, P.A.
A Theoretical Basis for the Synthesis and Verification of Systolic Designs.
Technical Report Internal Report, Dept. of Computer Science, University of Utah, June, 1982.

[45] Subrahmanyam, P.A.
On Automating the Computation of Approximate, Concrete, and Asymptotic Complexity Measures of VLSI Designs (to appear).
Dept. of Computer Science, University of Utah, November, 1982.

[46] Subrahmanyam, P.A.
Automatable Paradigms for Software-Hardware Design: Language Issues.
In J.Rader (editor), IEEE Workshop on VLSI and Software Engineering. IEEE, October, 1982.
Also available as University of Utah Technical Report UTEC-82-096, September 1982.

[47] Subrahmanyam, P.A.
An Algebraic Basis for VLSI Design.
Draft of a Research Monograph, April 1982, 120 pp. Available from the Department of Computer Science, University of Utah.

[48] Subrahmanyam, P.A.
An Automatic/Interactive Software Development System: Formal Basis and Design.
North-Holland, Amsterdam, 1982, .

[49] Subrahmanyam, P.A.
Synthesizing VLSI Circuits from Behavioral Specifications: A Very High Level Silicon Compiler and its Theoretical Basis.
In F. Anceau (editor), VLSI 83, International Conference. North Holland, 1983.

[50] Purushothaman, S., and Subrahmanyam, P.A.
An Algebraic Basis for Specifying and Reasoning about Protocols For Designing Self Timed Circuits.
In F. Anceau (editor), VLSI 83, International Conference. North Holland, 1983.

[51] Subrahmanyam, P.A.
Overview of a Conceptual and Formal Basis for an Automatable High Level

Design Paradigm for Integrated Systems.
In Proceedings of the IEEE International Conference on Computer Design and VLSI. IEEE, October, 1983.
Summary of an invited talk presented at ICCD, 1983.

[52] Subrahmanyam, P.A. and You, J-H.
Pattern Driven Lazy Reduction: A Unifying Evaluation Mechanism for Functional and Logic Programs.
In Proceedings of the Eleventh Annual ACM Symposium on Principles of Programming Languages. ACM, Jan, 1984.
To appear.

[53] Subrahmanyam, P.A. and You, J-H.
Conceptual Basis and Evaluation Strategies for a Programming Language Integrating Functional and Logic Programming.
In Proc. 1984 International Symposium on Logic Programming. Feb., 1984.
To appear.

[54] Jiro Tanaka.
Optimized Concurrent Execution of an Applicative Language.
PhD thesis, University of Utah, 1983.
To appear.

References

[1] Carter, T. M., Davis, A., Hayes, A. B., Lindstrom, G., Klass, D., Maloney, M. P., Nelson, B. E., Organick, E. I. and Smith, K. F.
Transforming an Ada Program Unit into Silicon and Verifying its Behavior in an Ada Environment: A First Experiment.
September, 1983.

[2] R. M. Keller and R. Sleep.
Applicative caching: programmer control of object sharing and lifetime in distributed implementations of applicative languages.
In Conf. on Functional Languages and Computer Architecture, pages 131-140. MIT, Oct., 1981.

[3] R.M. Keller and F. Lin.
The Rediflow Simulator
1983.

[4] R.M. Keller.
Rediflow Multiprocessing.
In Compcon '84. IEEE, 1984.

[5] F. Lin.
A Distributed Load Balancing Mechanism for Applicative Systems.
1983.

[6] G. Lindstrom, P. Panangaden.
Stream-Based Execution of Logic Programs.
To appear in Proc. 1984 Int'l. Symp. on Logic Programming, Atlantic City, Feb. 1984.

[7] P. Mishra and R. M. Keller.
Abstract interpretation of Applicative programs.
In Eleventh Annual Symposium on Principles of Programming Languages. ACM, January, 1984.

[8] P. Mishra.
Polymorphic Type Inference in Prolog.
In 1984 International Symposium on Logic Programming. IEEE, February, 1984.

[9] Organick, E.I.
Signal Processing-to-Silicon Using Ada as a Hardware Specification Language: An Initial Investigation.
August, 1983.
Internal Memorandum, Dept. of Computer Science, University of Utah.

[10] Piloty, R. Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F., Skelly, P.
CONLAN Report.
Springer-Verlag, Berlin, 1983.
Lecture Notes in Computer Science. Vol 151, Goos and Hartmanis, Editors.

[11] Preston, G.W.
Report of IDA Summer Study on Hardware Description Language.
Technical Report IDA Paper P-1595, Institute for Defense Analysis, Science and Technology Division, Oct., 1981.

[12] Rajopadhye, S. and P.A. Subrahmanyam.
Formal Semantics for a Symbolic IC Design Technique: Examples and Applications.
Technical Report UTEC # 83-041, University of Utah, May, 1983.

[13] Ramakrishnan, I.V. and P.A. Subrahmanyam.
On Mapping Hypercube Graphs onto Linear Systolic Arrays.
Technical Report CS-83-203, University of Utah, March, 1983.

[14] U.S. Reddy.
Inductive Basis: An Abstraction Mechanism for Equational Programs.
1983.

[15] U.S. Reddy.
Transforming Logic Programs into Functional Programs.
In 1984 International Symposium on Logic Programming. IEEE, February, 1984.

[16] Subrahmanyam, P.A.
Abstractions to Silicon: A New Design Paradigm for Special Purpose VLSI Systems.
Technical Report UTEC # 82-065, University of Utah, January, 1981 (Revised May 1982).
Submitted for Publication to TOCS.

[17] Jiro Tanaka and Robert M. Keller.
S-code Extension in Function-Equation Language: User's Manual
Department of Computer Science, University of Utah, 1983.
unpublished memo.

[18] Jiro Tanaka.
Optimized Concurrent Execution of an Applicative Language.
PhD thesis, University of Utah, 1983.
to appear.